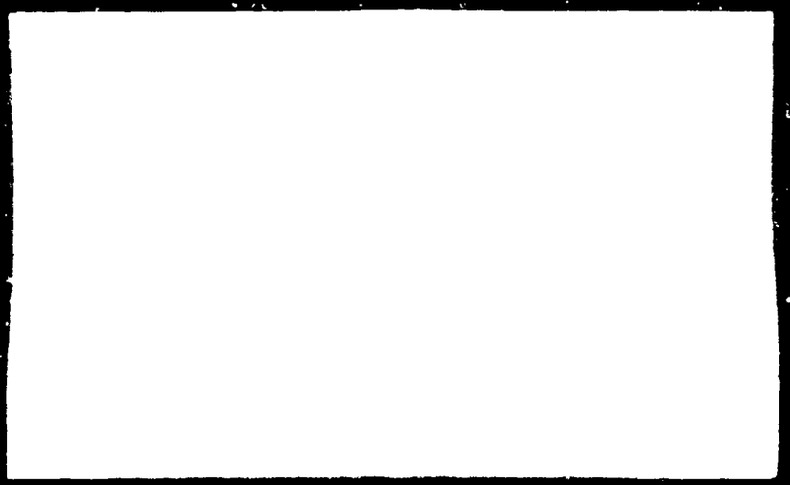
LEVEL II

AD A108824



DTIC
ELECTE
DEC 23 1981
S D
D

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

UNBOUNDED SPEED VARIABILITY IN DISTRIBUTED COMMUNCIATIONS SYSTEMS

John H. Reif*
Paul G. Spirakis

TR-14-81

*This work was supported in part by the National Science Foundation Grant NSF-MCS79-21024, and the Office of Naval Research Contract N00014-80-C-0647.

Unclassified

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

REPORT DOCUMENTATION PAGE	READ INSTRUCTIONS BEFORE COMPLETING FORM
1. REPORT NUMBER	2. GOVT ACCESSION NO. AD-A108824 / 3. RECIPIENT'S CATALOG NUMBER
4. TITLE (and Subtitle) Unbounded Speed Variability in Distributed Communications Systems	5. TYPE OF REPORT & PERIOD COVERED Technical Report
	6. PERFORMING ORG. REPORT NUMBER
7. AUTHOR(s) John H. Reif, Paul G. Spirakis	8. CONTRACT OR GRANT NUMBER(s) N00014-80-C-0647
9. PERFORMING ORGANIZATION NAME AND ADDRESS Harvard University, Cambridge, MA 02138	10. PROGRAM ELEMENT, PROJECT, TASK AREA & WORK UNIT NUMBERS
11. CONTROLLING OFFICE NAME AND ADDRESS Office of Naval Research, 800 North Quincy Street, Arlington, VA 22217	12. REPORT DATE 1981
	13. NUMBER OF PAGES 15
14. MONITORING AGENCY NAME & ADDRESS (if different from Controlling Office) same as above	15. SECURITY CLASS. (of this report)
	15a. DECLASSIFICATION/DOWNGRADING SCHEDULE

16. DISTRIBUTION STATEMENT (of this Report)

unlimited

DISTRIBUTION STATEMENT A
Approved for public release;
Distribution Unlimited

17. DISTRIBUTION STATEMENT (of the abstract entered in Block 20, if different from Report)

unlimited

18. SUPPLEMENTARY NOTES

19. KEY WORDS (Continue on reverse side if necessary and identify by block number)

real time, synchronization, parallel algorithm, distributed communication, CSP, multiprocessing

20. ABSTRACT (Continue on reverse side if necessary and identify by block number)

See reverse

81 12 22 110

DD FORM 1 JAN 73 1473 EDITION OF 1 NOV 65 IS OBSOLETE
S/N 0102-014-6601

SECURITY CLASSIFICATION OF THIS PAGE (When Data Entered)

20 .

Abstract

This paper concerns the fundamental problem of synchronizing communication between distributed processes whose speeds (steps per real time unit) vary dynamically. Communication must be established in matching pairs, which are mutually willing to communicate. We show how to implement a distributed local scheduler to find these pairs. The only means of synchronization are boolean "flag" variables, each of which can be written by only one process and read by at most one other process. (Shared variables are very difficult to implement in the case the processes are running in different processors in a communication network.) No global bounds in the speeds of processes are assumed. Processes with speed zero are considered dead. However, when their speed is nonzero then they execute their programs correctly. Dead processes do not harm our algorithms' performance with respect to pairs of other running processes. When the rate of change of the ratio of speeds of neighbour processes (i.e. relative acceleration) is bounded, then any two of these processes will establish communication within a constant number of steps of the slowest process with high likelihood. So, our implementation has the property of achieving relative real time response. We can use our techniques to solve other problems such as resource allocation and implementation of parallel languages such as CSP and Ada. Note that we do not have any probability assumptions about the system behavior, although our algorithms use the technique of probabilistic choice.

UNBOUNDED SPEED VARIABILITY IN DISTRIBUTED COMMUNICATION SYSTEMS

by

John Reif and Paul Spirakis
Aiken Computation Laboratory
Division of Applied Sciences
Harvard University, Cambridge, Massachusetts 02138

1.1 Abstract

This paper concerns the fundamental problem of synchronizing communication between distributed processes whose speeds (steps per real time unit) vary dynamically. Communication must be established in matching pairs, which are mutually willing to communicate. We show how to implement a distributed local scheduler to find these pairs. The only means of synchronization are boolean "flag" variables, each of which can be written by only one process and read by at most one other process. (Shared variables are very difficult to implement in the case the processes are running in different processors in a communication network.) No global bounds in the speeds of processes are assumed. Processes with speed zero are considered dead. However, when their speed is nonzero then they execute their programs correctly. Dead processes do not harm our algorithms' performance with respect to pairs of other running processes. When the rate of change of the ratio of speeds of neighbour processes (i.e. relative acceleration) is bounded, then any two of these processes will establish communication within a constant number of steps of the slowest process with high likelihood. So, our implementation has the property of achieving relative real time response. We can use our techniques to solve other problems such as resource allocation and implementation of parallel languages such as CSP and Ada. Note that we do not have any probability assumptions about the system behavior, although our algorithms use the technique of probabilistic choice.

1.2 Introduction

Recently, [Reif, Spirakis, 1981] showed how to achieve real-time response using probabilistic synchronization techniques, with the assumption that the speeds of all processes were bounded between fixed nonzero bounds. This lead to (see appendices I and II of [Reif, Spirakis, 1981]) real time resource allocation algorithms and real time implementation of message passing in CSP.

In this paper we assume *no global bounds on the processors' speeds*. They can vary dynamically from zero to an upper bound which may be different for each processor, and not known by the other processors. We allow a possibly infinite number of processes, so that there may not be a global upper bound on the speeds. Processes may die (have zero speed) but when they have nonzero speed then we assume they execute their programs correctly. We are interested in *direct* interprocess communication (rather than packet switching) which is of the form of *handshake* (rather than buffered), as in Hoare's CSP. [Hoare, 1978]. The essential technique that we utilize is that of *probabilistic choice*. This technique, introduced to synchronization problems by [Rabin, 1980], [Lehmann and Rabin, 1981] and [Francez, Rodeh, 1980], was also utilized in our previous work.

The use of probabilistic choice in the algorithms leads to considerable improvements in the space and time efficiency [Rabin, 1980], [Reif, Spirakis, 1981]; we feel that this may be because complex sequences of processes' steps prohibiting communication have very low probability of occurrence. We also introduce new *adaptive techniques:* The processes estimate the speeds of neighbour processes and select them to communicate with probabilities depending on the speeds, penalizing the slowest processes. These adaptive techniques do not seem to have ever been utilized in the previous synchronization literature.

1.3 Relative real time response

If processes are bounded in speed then it is natural to define real time response to be a response to a communication request that uses no more than constant number of units of real time. This measure is inapplicable in our case in which there is no global upper bound and no nonzero lower bound on speeds. Thus we introduce the notion of *relative real time response* which is establishment of communication between *any pair* of neighbouring processes within constant number of local rounds (A *local round* of neighbour processes, i,j is the minimum time interval which contains at least one step of each process and exactly one step of the slower of i,j). We achieve this by our probabilistic algorithms with some probability of error which can be made arbitrarily low. We conjecture that it is not possible to achieve relative real time response without use of randomization. The best *deterministic* symmetric algorithms which attempt to form matchings in distributed systems have a relative response depending linearly on the network's diameter. (Also, [Arjomandi, Fischer, Lynch, 1981] have actually shown that some synchronization problems which are global (in contrast to our problem) cannot be done in real time and require time proportional to the logarithm of the total number of processors in the network. A typical situation where this could occur is the problem of detecting connected components of processes whose speeds are within given bounds e.g., with nonzero values.)

2. The model VSDCS (Variable Speed Distributed Communication System).

We develop here a theoretical model related to, but more general than, the DCS (Distributed Communication System) of [Reif, Spirakis, 1981]. We will attempt to describe our model in words and avoid mathematical notation in this abstracted draft. A detailed description of the fundamental issues can be found in [Reif, Spirakis, 1981].

We assume a possibly infinite collection of processes $\pi = \{1,2,\ldots\}$. Events of the system are totally ordered on the real-time line $[0,\infty)$. Each process consists of a fixed set of synchronous parallel subprocesses (i.e. with same speeds) where we distinguish the *director* subprocess. The director wishes at various times to communicate with directors of other processes but has no means of communication except via the communications system. This is implemented by many *poller* subprocesses (three for each target process) which are synchronous with themselves and the director. We assume a fixed connections graph H which is undirected and has the set π as its vertex set. An edge $\{i,j\}$ indicates that process i is physically able to communicate with process j (but not necessarily willing to). H is assumed to have finite valence. We also assume for each time t the *willingness digraph* G_t which

indicates the willingness of a given process i to communicate with a given neighbour j at a given time t. (We indicate it by the edge $i \xrightarrow[t]{} j$ and say i is a *willing neighbour* of j.).

Note that $i \xrightarrow[t]{} j$ only if $\{i,j\} \in H$

The edges of this graph are stored distributedly so that the edges departing from a given process are only known to that process. We assume that the outdegree of each vertex of G_t is upper bounded by a fixed constant v. For each $t \geq 0$, the (possibly infinite) digraph M_t with vertices π and directed edges $i \xrightarrow[t]{} j$ denotes which processes *open communication* to other processes at time t. We denote $i \xleftrightarrow[t]{} j$ if both $i \xrightarrow[t]{} j$ and $j \xrightarrow[t]{} i$. Thus $i \xleftrightarrow[t]{} j$ denotes i,j *achieve mutual communication* at time t. M_t is the digraph that implementations of distributed synchronization achieve. We wish implementations to be *proper* in the sense that

(a) $i \xrightarrow[t]{} j$ only if $i \xleftrightarrow[t]{} j$ (neighbours try to speak only if they are mutually willing to)

(b) $\xleftrightarrow[t]{}$ must be a partial matching.

If $i \xleftrightarrow[t]{} j$ then not $j' \xleftrightarrow[t]{} i$ for any $j' \in \pi - \{j\}$. (Nobody is allowed to speak with two or more neighbours at the same time.).

Again, we assume that processes can suddenly die (i.e. have zero speeds) but when they are awake they execute their programs correctly. We furthermore assume that each process has an upper bound on its speed which may be different from the other processes and not known to them.

We define the *relative acceleration bound* α of processes i and j to be the worst case absolute rate of change of the ratio of steps of the two processes per step of any of the two processes. The correctness of our synchronization algorithms does not depend on whether processes are acceleration bounded, however we assume fixed acceleration bound α in our time complexity analysis. (i.e. the relative acceleration of one neighbour with respect to another is bounded by a constant bound α or can be $-\infty$ if the process dies).

2.2 Communication commands for the VS-DCS

The following communication primitives can be implemented by the poller subsystem and executed by the directors of each process: (The director may not get an immediate answer but may proceed to some other instruction and later a time slot for communication will be arranged by the poller).

ATTEMPT-COM$_i(p_j)$: indicates that the director of i wishes to communicate with the director of process j.

CANCEL-COM$_i(p_j)$: indicates that the director of i wishes no longer to communicate with p_j.

The precise semantics of ATTEMPT-COM and CANCEL-COM are given by the relation $\xrightarrow[t]{} \subseteq \pi \times \pi$ (the willingness digraph, defined in section 2.1).

2.3 Complexity of VS-DCS

We assume a worst-case oracle $\mathcal{A}$ which at time 0 chooses the speeds of all the processes for all times. $\mathcal{A}$ is also able to dynamically change the willingness relation $\xrightarrow[t]{}$ (i.e. dynamically choose one of ATTEMPT-COM, CANCEL-COM for the directors to execute) so as to achieve the worst case performance of the implementation of VS-DCS

We say $\mathcal{A}$ *is tame for* i, j on time interval Δ if the pairs $\{(i,j)\} \cup \{(i,k) \mid k \text{ is a neighbour of } i\} \cup \{(j,k) \mid k \text{ is a neighbour of } j\}$ are each relative acceleration bounded by α on the time interval Δ. (We assume here a global constant α).

For every ε on (0,1) let the ε-*error response* $S(\varepsilon)$ be an integer > 0 such that for every pair of neighbours i,j and each time interval Δ and for every oracle $\mathcal{A}$ which is tame for i,j on Δ, if $i \xleftrightarrow[\Delta]{} j$ and the number of steps of the slowest of i,j within Δ is $\geq S(\varepsilon)$ then there exists a subinterval $\Delta' \subseteq \Delta$ such that $i \xleftrightsquigarrow[\Delta']{} j$ with probability $\geq 1-\varepsilon$. Intuitively $1-\varepsilon$ gives a lower bound in the probability of establishing communication in the case process i issues an ATTEMPT-COM (p_j) at the beginning of Δ, and after $S(\varepsilon)$ steps it calls CANCEL-COM (p_j). (Note that we presume here that i and j and their neighbours have relative acceleration bound α during the interval Δ; at other times this acceleration bound may be violated, and furthermore the acceleration bound α need not hold for other processes even during the interval Δ).

We consider an implementation of VS-DCS to be *relative real time* if for all constants ε on (0,1), the relative ε-error response $S(\varepsilon)$ is independent of any global measure of the willingness digraph G_t (such as $|\pi|$ or any function of it) and independent of any local measure of G_t except the constant maximum valence v of the vertices of G_t. ($S(\varepsilon)$ may depend on the bound α on the relative acceleration). Note that relative real time response *does not imply* that communication is guaranteed within any time interval but instead it is guaranteed within a bounded number of steps of the processes with high likelihood (this is because processes can slow down arbitrarily). In this paper we show how to implement the VS-DCS so that relative real time response is achieved.

3. Applications of VS-DCS

The primitives ATTEMPT-COM, CANCEL-COM of VSDCS are powerful enough to supply real time implementations of synchronization constructs of high-level parallel languages like CSP and Ada.

3.1 Real time resource granting systems with process failures

Previously, in [Reif, Spirakis, 1981] we utilized the more restricted DCS system (which does not allow process failures) to implement a real time resource granting system. In this paper, we can cope with sudden process failures (zero speeds). In this case, the process governing a resource will attempt to communicate for $\alpha s^2(\epsilon)/2$ of its steps with a process granted the resource. If that process does not respond, the resource governing process may reclaim the resource. If a resource allocator dies, then other processes can play its role.

3.2 Relative real time implementation of CSP and Ada's synchronization constructs.

In a typical stage during execution, the processes comprising a CSP program may be divided into two classes: those busy with local computations and those waiting for a partner to communicate with. A distributed guard scheduler can be implemented by using the poller subprocesses of the relative real time VS-DCS system.

Also, in Ada, two-way communication between pairs of tasks is allowed in synchronized time instances called *rendezvous*. An *accept* statement of the form *accept* f(-) appearing in task T_1 indicates that T_1 is willing to rendezvous at f with any task of similar argument type. The task T_2 may execute a *call* statement of the form f(-) indicating that T_2 is willing to *rendezvous* with T_1 at the accept statement containing f. Ada also allows for *selective accept statements* containing multiple accept statements, one of which must be nondeterministically chosen to execute. (This is similar to the *select* statement of CSP).

Ada's tasks may be implemented by processes whose speeds vary dynamically. (Processes may even fail for various time intervals.) The key implementation problem is to synchronize task rendezvous within relative real time, in spite of the dynamic speed variations. These processes may be connected within a distributed network whose transmission channels may also have variable speeds or fail. Unreliable transmission channels can be viewed as processes which are connected with the processes of the network via reliable communication channels.

We assume that it is possible to analyze (perhaps by data flow analysis) an Ada program to determine an indirected (possibly infinite) connections graph whose nodes are all the tasks possibly created by the Ada program and edges are

the possible task communication pairs. Since an actual implementation will have in its hands at any time only a finite set of processes we assume that only the currently active tasks have an associated implementing process and that a call to Ada's *initiate* statement devotes a currently free process to a given newly created task. An *abort* statement garbage collects the implementing process from the deleted task and places it back to the free list of processes. These implementation techniques were developed by [Denis and Misunas, 1974] for real time implementation of data flow machines.

The synchronization facilities of the VS-DCS system provide a real-time implementation of the *accept* and *call* statements. A version of the *active* statement can be implemented so that deleted tasks can be detected by their neighbours in real time with some (arbitrarily small) error probability. Finally, the *symmetry* and *locality* of the VS-DCS implementation (due to its probabilistic nature) may help in eliminating the tradeoff between generality of expression and ease of implementation in Ada.

The *probabilistic fairness* guaranteed by the algorithms of the pollers eliminates the danger of bottlenecks which could be created if conventional techniques were used (a new task which centralizes requests and keeps track of busy server tasks is one of the conventional proposed solutions). Most of the problems which VS-DCS could cure are discussed in [Mahjoub, 1981], [Francez, Rodeh, 1980]. A probabilistic solution to some of the discussed problems was given also in [Francez,Rodeh, 1980] but no discussion about real-time properties was done and neither the problem of speed variations and dying processes was addressed.

4. Relative Real Time Implementation of VS-DCS

4.1 Intuitive description of the algorithm.

We utilize $3v + 1$ synchronized parallel processes to implement the poller subprocess for each process i. These are the *communicators* $p_1^i, p_2^i, \ldots, p_{2v}^i$, the *speed estimators* $e_1^i, \ldots, e_v^i$ and the *judge* subprocess of process i. Each pair of the communicators $p_k^i, p_{k''}^i$ (with $k' \bmod v = k'' \bmod v = k$) is devoted to communication with a specific neighbour (the k^{th} neighbour). Each estimator is used to continuously update an estimation of the speed of a particular neighbour process. The judge has the task to select under certain conditions one communicator and to give to him the right to open the communications channel of node i to its corresponding neighbour. We frequently use the technique of *handshake* by which we mean that each subprocess modifies a flag variable observed by the corresponding neighbour subprocess. Process contention between synchronized processes is easy to implement (we can allow each to take a separate step in a small round).

Our algorithm for the k^{th} communicator subprocess p_k^i $(1 \leq k \leq 2v)$ of the poller of process i proceeds as follows:

Let $k' = k \bmod v$. At every time $t \geq 0$, $E_i(1), \ldots, E_i(D_i)$ is the list of targets of edges of G_t departing from $i \in \pi$, and D_i is the current number of the targets ($D_i \leq v$). Those variables are dynamically set by the oracle $\mathcal{A}$ and they are the neighbours to which process i is willing to open communication at time t. The subprocess p_k^i deals with the $E_i(k')$ neighbour. If $k \leq v$, then p_k^i is an *asker* subprocess, else it is a *responder* process. p_k^i must first handshake with the corresponding subprocess of process $E_i(k')$ to which node i wishes to communicate. We need two handshake subprocesses (ask, respond respectively) per neighbour because of a certain asymmetry in the handshake (some has first to modify a flag). In particular the asker procedure initiates the handshake and the responder answers to it.

Next we wish to find a time slot in which the two neighbours may communicate. Because there may be contention among other processes j which also wish to communicate with i (and consequently, other askers or responders of node i also will handshake) we must resolve the contention by a fair judge. To do this, we add the process p_k^i to a queue and an additional synchronous subprocess of poller i, the *judge* takes a random process from this queue and allocates time slots for communication attempts. To ensure that slower neighbours do not utilize any more total time on the average than faster neighbours during communication attempts, we weigh the probabilities of subprocesses to be chosen from the queue by the factor

$$\frac{1/\Delta_{ik}}{\sum_j 1/\Delta_{ij}}$$

where Δ_{im} ($m = 1, \ldots, v$) is the current estimation of the steps of process i per step of process m, supplied by the estimator e_m^i.

This has the effect that each subprocess in the queue attempts to communicate on the average $1/2v$ of the total time. (See the analysis for a proof of that). If a process is chosen by the judge but the communication is not established, the algorithm requires that subprocess to initiate another handshake with its partner (to check if they are still mutually willing to communicate and to synchronize steps). Then, it is again added to the queue to be given another chance to establish communication. This process proceeds until either the director of i withdraws its willingness to communicate with $E_i(k')$ or until it establishes

communication. Note that the slots allocated by the judge to each selected communicator, take into account the current speed ratio of the node i and its neighbour corresponding to that communicator, adjusted by a factor related to worst-case acceleration, to give the opportunity of at least one step overlap in time of process i and its neighbour, if their corresponding channels are both open.

We introduce random waits which help subprocess p_k^i to eliminate the possibility of schedules set-up by the oracle $\mathcal{A}$ to have always a particular subprocess arrive first in the queue and win the contest. This possibility is eliminated since the oracle sets the speeds at time 0 and cannot affect the random choices done by the processes.

Also, our algorithms assume each process has a perfect random bit generator, independent of the random bits generated by other processes.

Note that we trade computation effort (parallelism) in a node to achieve reliable communication. This parallelism is limited because of the bounded valence v of the graph G_t. We can always simulate these synchronous parallel subprocesses in a node i by a single processor, using round robin techniques. This will reduce the effective speed of each subprocess by only a factor of $3v + 1$.

4.2 The programs of the pollers

For each poller $i \in \pi$, we assume

Synchronous subprocesses

$p_1^i, p_2^i, \ldots, p_v^i$ (askers)

$p_{v+1}^i, \ldots, p_{2v}^i$ (responders)

$e_1^i, \ldots, e_v^i$ (estimators)

and the $judge_i$

Each of the communicators p_k^i executes the following program

```
process  p_k^i
        locals   Δ_ik
WHILE   true  DO
        LO:  IF   D_i ≥ k  THEN
             BEGIN .
        L1:  Choose  W  at random from [0,4(2v+1)(2α+1)]
             do  W  noops
             if  k ≤ v  then  ASK(E_i(k)) else  RESPOND(E_i(k))
             add  k  to  Q
             WHILE  marriage_i(k)=0  DO  noop
                    x ← ESTABLISH-COM (E_i(k),4(2 α + 1)· Δ_ik)
                    marriage_i(k) ← 0
                    IF  x  THEN GO TO  L1  ELSE GO TO LO
             END
OD
```

The speed estimator e_k^i:

```
process   e_k^i
DO        FOREVER
          set F_ik to 1.
          wait until F_ki is set
A:        zero F_ik ; s ← CURSTEP
          wait until F_ki is zeroed
B:        Δ_ik ← (CURSTEP-s)/2
OD
```

Note that F_{ik} is a flag set by i, read by k.

The special register CURSTEP gives the current step of process i. We assume that a step consists of an elementary statement of the program; e_k^i's execution assures that Δ_{ik} is (within a factor of 2) the actual speed ratio of processes i and k, since from step A to step B the fastest of the partners does CURSTEP-s steps and the slowest does 2 steps

```
process   judge_i
  WHILE   true DO
          IF   Q ≠ ∅   THEN

               BEGIN
               choose and delete a random element k of the queue
               Q with probability  1/Δ_ik / Σ_j 1/Δ_ij

               marriage_i(k) ← 1
               WHILE marriage_i(k) = 1 DO noop
               END

          END
OD
```

Note: The assignment of the $marriage_i$ variable by the judge allows p_k^i to attempt a communication with the corresponding subprocess of the neighbour $E_j(k')$.

The following are the low level synchronization procedures used by the poller programs:

```
procedure      ask_i (target)
    BEGIN
        Q_i,target ← 1;
        WHILE A_target,i = 0  DO  noop

        Q_i,target ← 0;
        WHILE A_target,i = 1  DO  noop

    END
```

Note: The set of the flag $Q_{i,target}$ means that i asks the target. If the target detects $Q_{i,target} = 1$ then it answers positively by setting $A_{target,i} = 1$. Both partners reset these flags to 0 at the end of procedures ask and respond.

```
procedure      respond_i (asker)
    BEGIN
      LOOP UNTIL  Q_asker,i = 1
    BEGIN
        A_i,asker ← 1;
      WHILE      Q_asker,i = 1  DO noop
        A_i,asker ← 0;
    END
END
```

We finally present the code for the procedure ESTABLISH-COM$_i$(target,ℓ). During its execution node i opens its channel to node target. A simple protocol (symmetric handshake) is then attempted to see if the neighbour responded to that communication attempt. If the protocol succeeds then node i is sure that node target also opened its channel and communication took place. Else, node i knows that the attempt failed.

```
procedure      ESTABLISH-COM_i  (target,ℓ)
    BEGIN
       open channel_i,target;  COM_i,target ← 1
       So ← CURSTEP
       b ← 1
       WHILE  (COM_target,i = 0)  or  (b = 1)  DO
                BEGIN
                IF  CURSTEP-So ≥ ℓ THEN  b ← 0
                END

       OD
```

IF ($COM_{target,i}$ = 1) AND (b = 1) THEN success ⟵ 1
ELSE success ⟵ 0

b ⟵ 0 ; $COM_{i,target}$ ⟵ 0
close $channel_{i,target}$
return (success)

END

Note: $COM_{i,target}$ is a flag of node i and $COM_{target,i}$ is a flag of node target. open $channel_{i,target}$ corresponds to the appearance of $i \overset{t}{\longleftrightarrow} target$ at the time of its execution. Also, ℓ is the maximum number of steps we are allowed to keep channel open before we fail.

5. Correctness properties of our proposed implementation and time analysis.

Lemma 5.1 A matching (with respect to the relation $\overset{t}{\longleftrightarrow}$) is guaranteed by the implementation.

Proof

In any time instant, only one of the subprocesses of any poller can have the marriage variable set and its channel open. So, the relation $\overset{t}{\longrightarrow}$ is one-one which means that $\overset{t}{\longleftrightarrow}$ can not be more than a matching.

Lemma 5.2 Death of a process does not affect the communication of other processes.

Proof

Death of process "target" at any time will only cause blocking of only one subprocess (p^i_{target}) per neighbour i of target. This does not disrupt the other subprocesses of the neighbours.

Lemma 5.3 Suppose that i,j start to be mutually willing to communicate at time t and continue to be willing for 5 local rounds. Then all four subprocesses $p^i_{j_1}$, $p^i_{j_2}$ and $p^j_{i_1}$, $p^j_{i_2}$ (with j_1 modv = j_2 modv = j and i_1 modv = i_2 modv = i) will arrive in the queues of i and j in a constant number (5) local rounds.

Proof

Note that at each time the slower of i,j will do only one step in the busy waits of procedures ask or respond. The result follows simply by counting the steps to be executed in each of the procedures. □

Let Δ_{ij} be the current estimation (within a factor of two) of the ratio of steps of i per step of j (estimated by i).

Definition 5.3 Let $h = \sum_{k=1}^{2v} \frac{1}{\Delta_{ik}}$

Definition 5.4 Let ρ_{ij} be the ratio $1/(h \cdot \Delta_{ij})$

In the following we assume that the oracle $\mathcal{A}$ is tame with respect to processes i,j in the time interval they attempt communication.

Definition 5.5 Let $\bar{S}_{ij}$ be the average number of steps that p_j^i does before it is selected to attempt communication, measured from the time it enters the queue.

Lemma 5.4 $\bar{S}_{ij} \leq 8v \cdot (2\alpha+1) \cdot \Delta_{ij}$ where Δ_{ij} is the most current estimation

Proof

Note that the probability to be chosen follows a geometric density $(1-\rho_{ij})^{h-1} \cdot \rho_{ij}$ where h = the number of selections done before p_j^i is selected. Each time p_j^i is not chosen, it waits in the queue for an average time bounded above by

$$\sum_{k=1}^{2v} c \cdot \Delta_{ik} \rho_{ik}$$

where $c = 4(2\alpha+1)$.

So

$$\bar{S}_{ij} \leq \sum_{h=1}^{\infty} \text{prob}\left(\begin{matrix}h \text{ unfortunate} \\ \text{selections for } p_j^i\end{matrix}\right) \cdot h \cdot \sum_{k=1}^{2v} c \cdot \Delta_{ik} \rho_{ik}$$

$$\leq \frac{1}{\rho_{ij}} \cdot \sum_{k=1}^{2v} c \cdot \Delta_{ik} \rho_{ik} = h\Delta_{ij} \sum_{k=1}^{2v} \frac{c}{h} = 2vc\Delta_{ij}$$

□

Theorem 5.1 Each subprocess expects to get the channel $\geq 1/2v$ of the time.

Proof

The average number of steps p_k^i gets the channel is

$$S_Q = \frac{1/\Delta_{ik}}{h} \cdot 4(2\alpha+1) \cdot \Delta_{ik} = \frac{4(2\alpha+1)}{h}$$

In the worst case of process i being the fastest and all neighbours slowing down with the same worst case acceleration α, S_Q is the same for all processes in the queue. The worst contention happens when all 2v subprocesses are there. Hence, in the worst case, if T is a time interval and T_Q is the subinterval of T in which p_k^i has the channel, then

$$\text{mean}\left(\frac{|T_Q|}{|T|}\right) \geq \frac{S_Q}{2vS_Q} = \frac{1}{2v}$$

□

Note that this justifies the use of the estimate $\frac{1/\Delta_{ik}}{\sum_j 1/\Delta_{ij}}$ as the probability to select subprocess p_k^i from the queue.

In the following we assume $1 \leq i,k' \leq v$ and $k = k' + v$. Thus $p_{k'}^i$ is the asker and p_i^k is the responder.

Lemma 5.5 The probability of instantaneous overlap of subprocesses p_k^i and $p_i^{k'}$ in their channels is $\geq \left(\frac{1}{2v}\right)^2$

Proof

By theorem 5.1 and by the fact that the random waits, done by the subprocesses before each return to the queue, cause the relative position of the time intervals during which channels are open to be random, not affected by the oracle $\mathcal{A}$. (Note that it is essential here that the waits are uniformly distributed in the interval whose length is the mean number of local rounds attempt communication and given by Theorem 5.3). □

Definition 5.6 Let *success in communication* be an overlap of open channels for at least one step of both processes i,k'.

Definition 5.7 A *phase* of subprocess p_k^i is a random wait, a handshake with p_i^k, a wait in queue and a communication attempt.

Theorem 5.2 The probability of success in communication in a phase of subprocess p_k^i is $\geq \frac{1}{2}\left(\frac{1}{2v}\right)^2$

Proof

When the subprocess p_k^i opens its channel, the number of steps done from the time of the last estimation of Δ_{ik} is at most $2\Delta_{ik}$ and hence, the new speed ratio can be $(2\alpha+1)\Delta_{ik}$ in the worst case, (in which, process i is the fastest and process k' slows down continuously with the maximum acceleration, so that process i does more and more steps per step of k'). In this case, a communication attempt of $4(2\alpha+1)\Delta_{ik}$ time slots guarantees that p_i^k will do at least 2 steps during the time p_k^i has its channel open. Because of the random relative position of these steps with respect to p_k^i's steps (due to random waits), Prob (length of overlap is ≥ 1 step given that there is an overlap) $\geq \frac{2-1}{2} = \frac{1}{2}$

Hence

Prob (there is an overlap and its length is ≥ 1 step of both processes) $\geq \frac{1}{2} \cdot \left(\frac{1}{2v}\right)^2$

by Lemma 5.3 □

Definition 5.8 Let $\gamma_{min} = \frac{1}{2}\left(\frac{1}{2v}\right)^2$.

Definition 5.9 Let $q_{ik}\left(h/\mathcal{A}\right)$ be the probability that it takes exactly h phases for poller subprocess p_k^i to communicate with $p_i^{k'}$.

Lemma 5.10 For any oracle $\mathcal{A}$, $q_{ik}\left(h/\mathcal{A}\right) \leq (1-\gamma_{min})^{h-1}$

Proof

It sufficies to observe that the process of p_k^i be answered by $p_i^{k'}$ is a geometric stochastic process with success probability bounded by $[\gamma_{min},1]$. By using the above lemma and known expressions for the mean and the tail of a geometric we get

Lemma 5.7 $\quad \text{mean}(h) \leq \frac{1}{(\gamma_{min})^2}$

Lemma 5.8 $\quad \forall \varepsilon,\ 0 < \varepsilon < 1,\ \text{Prob}\{h > h_{max}(\varepsilon)\} \leq \varepsilon$

where

$$h_{max}(\varepsilon) = \frac{\log(\gamma_{min}\varepsilon)}{\log(1-\gamma_{min})}$$

Note that, in the worst case relation of speeds of processes i,k, the total length of a phase of subprocess p_k^i is the number of local rounds in the random wait plus the number of *local rounds* up to the end of the communication attempt, which is $8(2v+1)(2\alpha+1)$ (by the algorithms and by Lemma 5.4 and Theorem 5.1.) Hence we get

Theorem 5.3 For the worst case oracle $\mathcal{A}$, the mean number M of local rounds to achieve communication is

$$M \leq 64v^4\ 8(2v+1)(2\alpha+1)$$

and the ε-error response $S(\varepsilon)$ of the presented implementation of VS-DCS is

$$S(\varepsilon) \leq 8(2v+1(2\alpha+1) \cdot h_{max}(\varepsilon)$$

or

$$M = O(v^5\alpha)$$

and $\quad S(\varepsilon) = O\left(v^3 \cdot \log\left(\frac{v}{\varepsilon}\right) \cdot \alpha\right)$

Proof

By previous remark and the fact that

$$h_{max}(\varepsilon) = \frac{\log(\varepsilon/8v^2)}{\log\left(1 - \frac{1}{8v^2}\right)} \rightarrow 16v^2\log\left(\frac{v}{\varepsilon}\right)$$

□

Conclusion

Since we have assumed global parameters α and v to be constant, by Theorem 5.3 our system has relative real time response. Our restrictions on processors rates are much less than in our previous paper [Reif, Spirakis, 1981]. Furthermore, our programs seem much more modular and simple in design, although we have utilized new adaptive techniques to deal with arbitrary speed variability.

References

Angluin, D., "Local and Global Properties in Networks of Processors," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Arjomandi, E., M. Fischer, and N. Lynch, "A Difference in Efficiency between Synchronous and Asynchronous Systems," *13 Annual Symposium on Theory of Computing*, April 1981.

Bernstein, A.J., "Output Guards and Nondeterminism in Communicating Sequential Processes," *ACM Trans. on Prog. Lang. and Systems*, Vol. 2, No. 2, April 1980, pp. 234-238.

Dennis, J.B. and D.P. Misunas, "A Preliminary Architecture for a basic data-flow processor," *Proc. of the 2nd Annual Symposium on Computer Architecture*, ACM, IEEE, 1974, pp. 126-132.

Francez, N. and Rodeh, "A Distributed Data Type Implemented by a Probabilistic Communication Scheme," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 373-379.

Hoare, C.A.R., "Communicating Sequential Processes," *Com. of ACM*, Vol. 21, No. 8, Aug. 1978, pp. 666-677.

Lehmann, D. and M. Rabin, "On the Advantages of Free Choice: A Symmetric and Fully Distributed Solution to the Dining Philosophers' Problem," to appear in *8th ACM Symposium on Principles of Program Languages*, Jan. 1981.

Lipton, R. and F.G. Sayward, "Response Time of Parallel Programs," Research Report #108, Dept. Computer Science, Yale Univ., June 1977.

Lynch, N.A., "Fast Allocation of Nearby Resources in a Distributed System," *12th Annual Symposium in Theory of Computing*, Los Angeles, California, April 1980, pp. 70-81.

Mahjoub, A., "Some comments on Ada as a Real-time Programming Language," to appear.

Rabin, M., "N-Process Synchronization by a 4 $\log_2$N-valued Shared Variable," *21st Annual Symposium on Foundations of Computer Science*, Syracuse, New York, Oct. 1980, pp. 407-410.

Rabin, M., "The Choice Coordination Problem," Mem. No. UCB/ERL M80/38, Electronics Research Lab., Univ. of California, Berkeley, Aug. 1980.

Reif, J.H. and Spirakis, P., "Distributed Algorithms for Synchronizing Interprocess Communication Within Real Time," *13th Annual ACM Symposium on Theory of Computation*, Wisconsin, 1981, pp. 133-145.

Schwartz, J., "Distributed Synchronization of Communicating Sequential Processes," DAI Research Report No. 56, Univ. of Edinburgh, 1980.

Tonag, S., "Deadlock and Livelock-Free Packet Switching Networks," *12th Annual Symposium on Theory of Computing*, Los Angeles, California, April 1980, pp. 82-93.

Valiant, L.G., "A Scheme for Fast Parallel Communication," Technical Report, Computer Science Dept., Edinburgh, Scotland, July 1980.